Pages dynamisées par filtrage
Une mise en œuvre sur un cédérom à but pédagogique
Revision: 1.12

Christian Queinnec — Hélène Giroire

Université Paris 6 — Pierre et Marie Curie
LIP6, 4 place Jussieu, 75252 Paris Cedex

{Christian.Queinnec,Helene.Giroire}@lip6.fr

RÉSUMÉ. *Afin de s'adapter aux étudiants, à leur préférences ou à ce qui, tout simplement, semble approprié compte-tenu de l'enseignement qu'ils suivent, une solution couramment prise est d'utiliser des pages dynamiques (*servlets, *ou* Java Server Pages*). Notre propos est de montrer que ce choix peut être utilement complété par une mécanique de filtrage.*
Cette démarche a été employée avec succès pour la réalisation d'un cédérom, distribué en guise de polycopié électronique accompagnant un cours sur le langage C en licence d'informatique.

ABSTRACT. *Dynamic pages may generate content appropriate to their readers. These pages may depend on expressed or implied students' preferences. This paper shows that filtering the result of dynamic pages offers a wide range of additional facilities.*
This technique has been put to use with benefit for a CDROM to support a course on the C programming language.

MOTS-CLÉS : *Cédérom pédagogique, pages dynamiques, filtrage, HTML structuré, XML, système adaptatif*

KEYWORDS: *Educational CDROM, dynamic pages, filtering, structured HTML, XML, adaptive system*

1. Introduction

En septembre 1999 et 2000, l'UFR d'informatique de Paris 6 a distribué à tous ses étudiants de second cycle, en guise de « polycopié », un cédérom (nommé VideoC [QUE 00, CAZ 00a]) accompagnant un enseignement de licence dédié au langage C. Ce cédérom est destiné au travail personnel de l'étudiant et comporte un « serveur » local procurant des pages au navigateur de l'étudiant (cf. figure 1). Ces pages peuvent être extraites du cédérom ou d'un serveur distant accessible par Internet. Le serveur est local car il fonctionne sur la machine de l'étudiant et lui permet de butiner les sites collectés sur le cédérom sans nécessiter de connexion à Internet. En revanche, si une connexion est disponible, les pages absentes ou obsolètes du cédérom peuvent être rapatriées dynamiquement.

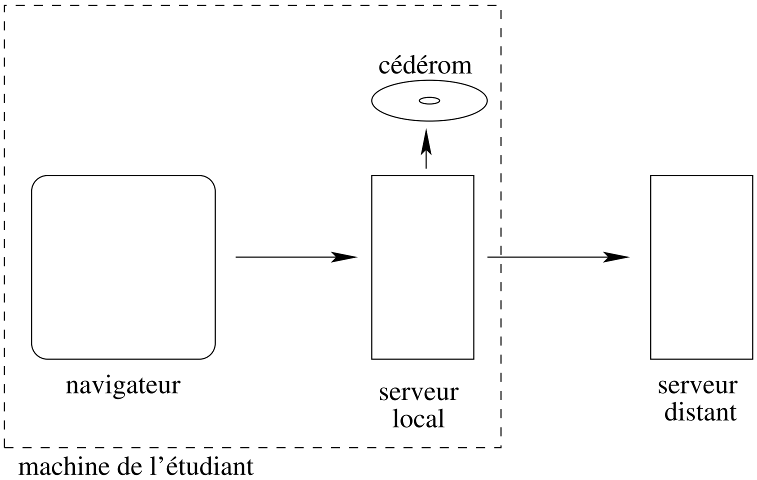

FIG. 1. *Organisation générale*

Le serveur local est le serveur Tomcat[1], un démon httpd similaire à Apache mais écrit en Java. Tomcat autorise *servlets* et *Java Server Pages*. Les *servlets* [COW 00] (resp. *Java Server Pages* [PEL 00] autrement dit JSP) permettent de créer programmatiquement en Java (resp. en un mélange de Java et d'HTML compilé vers une *servlet* Java) des pages actives dont le contenu est calculé lorsque la page est demandée. Ce calcul peut s'appuyer sur toutes les données recueillies auprès de l'étudiant et peut donc s'adapter aux préférences qu'il a pu exprimer, à sa progression, à ses résultats tels qu'ils ont pu être mesurés, etc.

Toutefois, nous avons fait en sorte que le résultat de ces pages (de l'HTML) passe par une couche finale de filtrage. Cette couche de filtrage permet de réaliser quelques effets intéressants comme :

1. `http://jakarta.apache.org/`

– insertion de bannière ou barre de menu standard,
– masquage ou remplacement de zones,
– insertion de liens de glossaires,
– l'adaptation du style de pages,
– l'ajout d'annotations situées.

2. Mécanique

Lorsque l'étudiant clique sur un lien, il requiert le contenu d'une page spécifiée par une URL. Cette URL est transmise par le navigateur de l'étudiant au serveur local qui l'analyse. En général, l'URL spécifie soit un fichier dont le contenu sera retourné à l'étudiant (texte HTML, images, sons, etc.), soit un programme qui sera exécuté et dont le résultat sera retourné à l'étudiant. Ce dernier mode correspond aux « pages dynamiques » puisqu'au contraire d'une page statique (un fichier), le contenu est calculé et peut donc dépendre de nombreux facteurs. *Servlets* et JSP (qui sont compilées en *servlets*) sont des exemples de pages dynamiques.

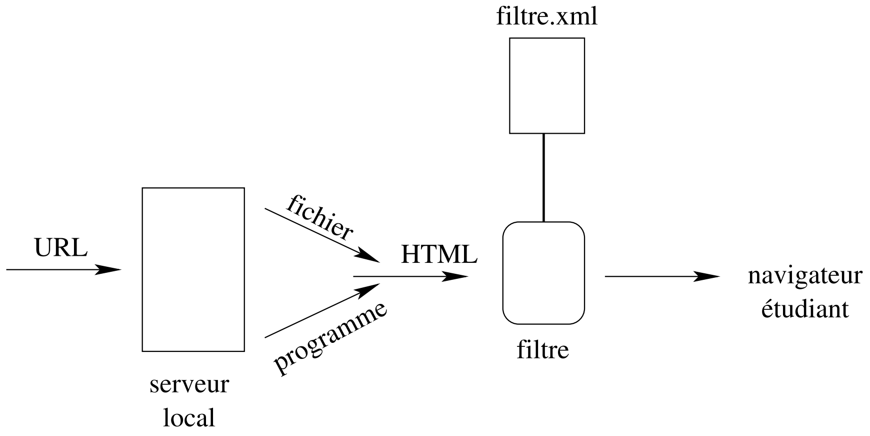

FIG. 2. *Flux d'informations*

Tous les contenus statiques ou dynamiques textuels en HTML passent au travers d'un filtre qui peut leur appliquer certaines transformations (cf. figure 2). On peut également imaginer que le filtre puisse traiter de données non textuelles et puisse, par exemple, changer couleurs ou tailles d'images pour s'adapter à une charte graphique.

La seconde version du serveur, celle de septembre 2000, a délocalisé une partie du serveur distant sur la machine de l'étudiant. Cela a notamment permis de totalement déporter la charge de filtrage vers les machines des étudiants au lieu de la concentrer sur un serveur unique partagé par tous les étudiants. Ce point avait posé problème l'an dernier lorsque 90 étudiants en TP demandaient en même temps les mêmes pages à un unique serveur devant filtrer individuellement chacune de ces pages.

Le processus de filtrage opère par réécriture d'HTML. Le contenu d'une page est matérialisé comme une grande chaîne de caractères en HTML. Le filtrage est décrit par une succession de règles. Chaque règle indique quelles sont les URLs qu'elle traite (sous la forme d'une expression régulière) et quels sont les fragments de texte à remplacer (là encore sous forme d'expressions régulières).

Les règles sont nommées, elles forment une séquence linéaire. Lorsqu'un texte doit être filtré, les règles lui sont appliquées tour à tour. Lorsqu'une règle s'applique (l'url convient), le contenu HTML est réécrit suivant la règle. Des attributs supplémentaires permettent de spécifier, si présents, la prochaine règle à appliquer ce qui permet de modifier partiellement l'ordre séquentiel immuable des règles. Ces attributs permettent (comme les étiquettes des règles pour l'outil sed [2] d'Unix) de court-circuiter la litanie des filtres à essayer pour, par exemple, cesser le filtrage dès qu'un filtre réussit.

Les règles sont spécifiées en XML [3] qui est un excellent format de données pour plusieurs raisons : *(i)* il est simple de paramétrer un analyseur syntaxique générique pour lire les règles, *(ii)* aucun cas d'erreur n'est à prévoir puisque les données ont une structure respectant une grammaire formellement spécifiée (une DTD en jargon), *(iii)* la bonne structuration des données peut être vérifiée avant qu'elles soient fournies au filtre, *(iv)* enfin, un outil graphique de création/modification de ces règles peut facilement produire ce type de données ce qui découple agréablement l'interface de l'essence.

Voici la DTD (pour *Document Type Definition*) décrivant les règles.

```
<?xml encoding="US-ASCII"?>
<!-- Describe how to filter URL content.
  $ Id: filter.dtd 1.2 2000/06/1 08:30:28 queinnec Exp $
  Copyright (C) 2000. Christian.Queinnec@lip6.fr

 This DTD should be used with the following DOCTYPE:
  !DOCTYPE filter PUBLIC
  "-//Christian Queinnec//DTD filter descriptor 1.2//EN"
  "http://youpou.lip6.fr/queinnec/DTD/filter.dtd"
 -->

<!ELEMENT filter (rule*)>

<!ELEMENT rule (path?,class,trigger?,description?,properties?)>
<!ATTLIST rule
          label CDATA #IMPLIED
>

<!-- No path means that the rule should always be applied -->
<!ELEMENT path (#PCDATA)>
```

2. http://www.ptug.org/sed/sedfaq.htm
3. http://www.w3.org/XML/

```
<!-- The fully qualified name of the Java class to run -->
<!ELEMENT class (#PCDATA)>

<!-- If present, that query parameter is mandatory in the URL -->
<!ELEMENT trigger (#PCDATA)>

<!-- A global comment on the rule -->
<!ELEMENT description (#PCDATA)>

<!-- The rule may use additional properties ie name-value pairs -->
<!ELEMENT properties (property*)>

<!ELEMENT property (name,value,meaning?)>

<!ELEMENT name (#PCDATA)>

<!ELEMENT value (#PCDATA)>

<!-- A comment on the property -->
<!ELEMENT meaning (#PCDATA)>

<!-- end of filter.dtd -->
```

Lorsque le serveur local est lancé, un filtreur (un composant) est lancé qui lit les règles qu'il doit appliquer depuis un fichier XML. On peut réinitialiser le filtreur en cours de route avec de nouvelles règles. Le contrôle du filtreur s'opère par des URLs particulières qu'interprète spécialement le serveur local.

3. Exemple

Voici, pour fixer les idées, un exemple concret d'une règle de filtrage :

```
<rule label="default">
  <path>/clic/.*\.html</path>
  <class>fr.lip6.qnc.videoc2000.filters.ReplacePart</class>
  <properties>
    <property>
      <name>regexp.to.change</name>
      <value>
<![CDATA[<!-- VideoC:Solution -->(.|\n)*?<!-- VideoC:EndOfSolution -->]]>
      </value></property>
    <property>
      <name>new.expression</name>
      <value><![CDATA[<div align=center>
<a href="<%current.url%>?solution=yes">Solution</a></div>]]>
      </value></property>
```

```
    <property>
      <name>query.parameter.name</name>
      <value>solution</value></property>
  </properties>
</rule>
```

Cette règle stipule que le serveur local lorsqu'il reçoit une URL demandant à lire un fichier du répertoire *clic* et de suffixe *html* doit la filtrer avec le filtre ReplacePart (écrit en Java). Ce filtre standard permet d'effectuer des substitutions spécifiées par des expressions régulières (avec une syntaxe à la Perl).

Un mécanisme général permet de paramétrer les filtres. Ce sont des propriétés dont on indique le nom et la valeur (correspondant aux balises name et value). Le filtre ReplacePart peut prendre plusieurs paramètres :

– le premier, regexp.to.change, est une expression régulière désignant les parties du texte sur lesquelles opérer.
– le second, new.expression, désigne le texte remplaçant le précédent. Si ce paramètre manque, le texte précédent est tout bonnement supprimé. On peut bien sûr utiliser des segments provenant du texte précédent pour construire le texte de remplacement.
– le troisième, query.parameter.name, désigne optionnellement, le nom par lequel débrayer le filtre au moment du filtrage. Une explication de ce dernier paramètre suit.

Les règles de filtrage sont statiquement exprimées en XML. Le fichier les décrivant est lu par le filtreur au moment de son initialisation. Tout contenu dont l'url est reconnue par une règle, doit donc subir cette règle ce que l'on veut parfois éviter. En conséquence, une URL comportant un paramètre de requête (pour *query parameter*) ayant pour nom le nom spécifié par query.parameter.name et une valeur booléenne, impose ou non que le filtrage ait lieu.

Ce mécanisme permet donc de forcer ou d'interdire qu'une règle particulière s'applique. La règle qui nous a servi d'exemple convertit le texte décrivant la solution d'un exercice en un lien vers cette solution. Tantôt on désire que la solution soit masquée, tantôt qu'elle apparaisse. Le lien menant vers la solution est un lien vers la page même que l'on filtre (tel qu'obtenu par le lexème `<%current.url%>` spécialement interprété par le filtre pour valoir l'URL de la page) mais avec un paramètre de requête `?solution=yes` indiquant que la solution est requise et que donc le filtre ne doit pas s'appliquer.

Pour résumer,
http ://127.0.0.1/clic/index.html est une URL qui sera filtrée,
http ://127.0.0.1/clic/index.html?solution=no le sera aussi
mais http ://127.0.0.1/clic/index.html?solution=yes ne sera pas filtrée.

4. Mise en œuvre dans VideoC

Dans cette section, nous montrons quelques exemples d'emploi du filtrage tel que mis en œuvre dans notre cédérom. Tout d'abord et parce que nous employions ce mécanisme, les pages qui sont servies ont une armature plus forte que celle usuellement requise par HTML. Nos pages sont écrites initialement en LaTeX, duquel nous dérivons des versions typographiées imprimables (en Postscript ou PDF pour des polycopiés classiques) ou des versions HTML (conversion réalisée par Hevea [MAR 99]) pour affichage sur navigateur. Nous pensons également, lorsque le passage à XML sera inéluctable, engendrer celui-ci à partir de nos fichiers LaTeX.

Les textes écrits en LaTeX sont composés suivant une armature arborescente proche (à la syntaxe près) de XML. Une note de cours est un environnement comportant des arguments tels que titre ou numéro unique d'identification de la notule ainsi que des sous-environnements (des champs) administratifs (auteur, mél, révision, etc.), des liens amont ou aval (c'est-à-dire requis ou suggérés) vers d'autres notules ou exercices, ainsi qu'un texte. Les exercices épousent la même armature sauf que le texte est lui-même composé d'énoncés, de questions et de solutions. Une solution est, par exemple, enserrée entre une commande de début d'environnement `\begin{answer}` et une commande de fin d'environnement `\end{answer}`.

Cette armature ne disparaît pas à la traduction en HTML, elle persiste sous forme de commentaires HTML indiquant début et fin des sections. On peut notamment le remarquer dans l'exemple précédent de règle : les solutions débutent par le commentaire HTML `<!-- VideoC:Solution -->` et s'achèvent par le commentaire HTML symétrique `<!-- VideoC:EndOfSolution -->`.

Plusieurs raisons ont motivé notre choix de rester en LaTeX plutôt que de passer à XML.

- Peu de navigateurs savent afficher du XML et notre cédérom devait être utilisable sous Linux et Windows.
- La traduction de XML en HTML doit souvent être effectuée sur le serveur ce que nous souhaitions éviter (pour les raisons de charges mentionnées plus haut).
- Comme montré plus haut, la structuration arborescente peut aussi bien être obtenue à partir de LaTeX qu'à partir de XML.
- Il est simple d'obtenir une édition soignée sur papier à partir de LaTeX. Seules les conversions XML vers LaTeX assurent ce même niveau de qualité [GOO 99].
- L'éditeur Emacs est très adapté à LaTeX.
- LaTeX est moins verbeux qu'XML.
- Les éditeurs d'XML sont rudimentaires et, comme tous les éditeurs structurels, ils sont plus adaptés à la maintenance (où il est primordial de ne pas perturber la structure du document modifié) qu'au développement ou à l'écriture spontanée de nouveaux textes. Le respect strict, séquentiel et continu d'une grammaire est incompatible avec la liberté nécessaire pour la conception d'un texte.
- Toutefois la structure d'un document XML peut être vérifiée ce qui est difficile à faire en LaTeX (notamment sur les expressions mathématiques).

Toute page possède en général plusieurs niveaux de lecture. Nos textes comportent donc des détails, typés, à deux niveaux. Un détail peut donc être une remarque d'implantation de niveau 1 ou une remarque syntaxique de niveau 2. Le texte normal, affiché par défaut, est considéré comme étant de niveau zéro. L'existence de détails sur la page déclenche l'insertion d'un menu supplémentaire indiquant leur présence et permettant de les obtenir. L'affichage peut s'effectuer par niveaux ou par nature de détails. Lorsque l'étudiant acquiert certaines capacités, comme par exemple d'avoir réussi un QCM d'implantation, certains niveaux de détails d'implantation peuvent alors être automatiquement présentés. On peut également imaginer pour un lecteur averti de lui laisser indiquer dans ses préferences s'il désire ou pas voir ces détails et à quel niveau.

Les détails et leurs niveaux persistent également dans l'HTML engendré toujours sous forme de commentaires HTML (ou de balises DIV ou SPAN).

Le choix de mettre beaucoup d'informations dans les pages nous a été dicté par la volonté de procurer un mode dégradé dit de secours. Si le serveur local tombe en panne ou que le cédérom est consulté sur une machine sur laquelle ne tourne pas le serveur local, alors l'étudiant n'a d'autre choix que de consulter directement les pages. Les pages doivent donc au minimum être lisibles, informatives et correctement agencées.

Cette volonté explique que dans une vaste majorité de cas, les filtrages ne sont que des masquages sélectifs d'information. Par défaut, nos pages cachent les informations administratives et tous les détails. Un bouton est inséré dans la barre de menu standard lorsque des détails sont présents afin qu'il soit possible de les obtenir.

5. Autres usages

Le masquage sélectif n'est pas la seule utilisation du filtrage, les filtrages peuvent également influencer l'apparence des pages.

Les liens amont ou aval peuvent être transformés en un menu déroulant ou une sous-fenêtre pour se conformer à un certain style de présentation graphique. Une barre de menu standard est, par exemple, insérée dans chacune des pages que nous produisons. La barre est synthétisée à partir d'une barre prototypique dont certains éléments ne sont démasqués qu'en fonction du contenu de la page (le bouton d'accès aux détails, le menu d'accès aux pages amont/aval, etc.) Les pages sont donc des réservoirs d'informations structurées, normalement visualisables par tout navigateur, à partir desquelles sont fabriquées de nouvelles pages spécialisées.

D'autres transformations sont possibles comme, par exemple dans notre cédérom, d'accommoder des pages que nous n'avons pas produites par nous-mêmes (des pages étrangères) mais que nous souhaitons utiliser. Par exemple, nous pouvons changer certaines balises HTML pour unifier le style des pages (couleur de fond (par ajout ou modification de l'attribut BGCOLOR de la balise BODY), feuille de style CSS (par insertion d'une balise LINK)). Nous pouvons aussi supprimer des parties comme les boutons de navigation (précédent, suivant, table des matières) qu'insèrent Hevea [4] [MAR 99] ou LaTeX2HTML [5]. Cette faculté de réemploi sans trop de heurt de pages que nous n'avons pas produites nous-mêmes est essentielle comme l'a prouvé notre cédérom.

Une autre transformation que nous sommes en train de mettre au point (il s'agit juste de développer le filtre adapté) consiste à analyser des programmes (C, Java, Scheme etc.) et à remplacer l'occurrence d'un identificateur par un lien vers la définition de celui-ci. Comme cela peut rendre la page assez chargée visuellement, le filtre est bien entendu débrayable. La même technique peut être employé avec des textes quelconques pour remplacer des mots par des liens vers un glossaire les expliquant. On peut ainsi (et là encore, cela nous semble essentiel) annoter des pages étrangères.

Enfin une dernière transformation que nous envisageons pour la prochaine rentrée sera de permettre l'annotation de pages par les étudiants. Un bout de code JavaScript permet aux étudiants de saisir un texte et d'indiquer où il doit être affiché sur la page (techniquement, l'annotation apparaît comme porté par un calque transparent au dessus du texte). Ces annotations sont alors transmises à un serveur global (le serveur d'annotations) de manière à pouvoir être partagées (entre tous les étudiants d'un TP ou d'une formation à distance). Lorsqu'une page doit être servie par le serveur local, le filtre interroge le serveur d'annotations et décore la page à servir avec les annotations associées. Ce mécanisme permet aux étudiants de pouvoir poser des questions situées par rapport au texte demandant des éclaircissements ou fournissant des explications. C'est une sorte de forum relatif à une certaine page, qui permet aux étudiants d'échanger des informations, aux enseignants de comprendre ce qui peut clocher dans un énoncé ou une solution, à tous de participer à l'édification de meilleures pages.

Les règles régissant un serveur local sont bien sûr propres au lieu où les étudiants travaillent. Les règles en salle de TP ne sont pas les mêmes que sur une machine non connectée à Internet dont ils disposent à domicile. Les annotations n'ont alors pas de sens.

6. Comparaisons

Le filtrage est une technique éprouvée et utile. La technique consistant à intercaler un relais (un « proxy ») entre son navigateur et le réseau permet souvent de transformer le contenu des pages en transit. Un tel relais comme, par exemple, Muffin[6] dû à Mark Boyns permet (depuis quelques années déjà) de supprimer des publicités ou d'insérer des liens vers un glossaire.

Le filtrage est si important que la nouvelle version de la norme des *servlets*, la version 2.3 [COW 00], l'introduit. Il est ainsi loisible de spécifier sur le serveur tournant les *servlets*, les filtrages à opérer soit avant, soit après.

4. `http://pauillac.inria.fr/hevea/`
5. `http://www-texdev.mpce.mq.edu.au/l2h/docs/manual/`
6. `http://muffin.doit.org`

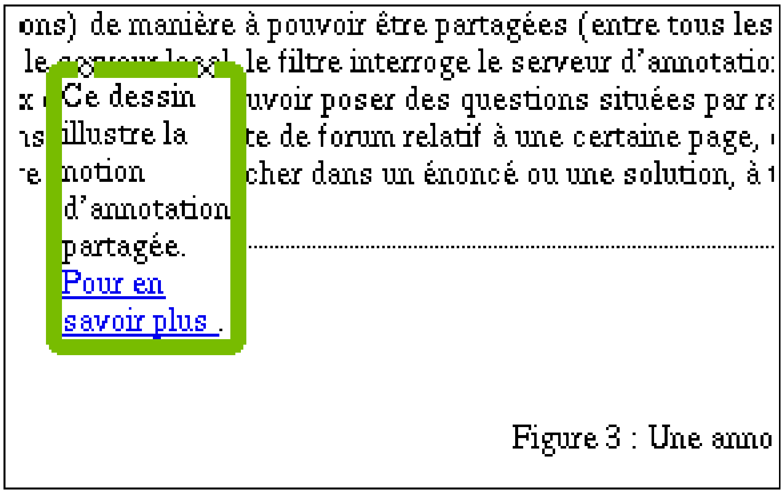

FIG. 3. *Une annotation partagée (vue d'artiste)*

Le filtrage permet de réaliser des effets décrits par certains auteurs comme les textes à trous de [MOU 00] ou les multiples activités de [SOM 00]. De même certaines formes d'adaptativité prônées en [BRU 96] peuvent être assurées par filtrage.

7. Conclusions

Bien que ne supposant que des techniques éprouvées, les effets que permet d'obtenir le filtrage couvrent un large spectre : simples masquages, réécritures, insertion de liens, modification de style, ajout d'annotations partagées, etc. Toutes sortes de transformation appréciées pédagogiquement.

Le filtrage que nous employons use d'expressions régulières plus amorphes mais moins limitées (car opérant sur un texte et non sur une structure arborescente) que les transformations spécifiées en XSLT à partir d'XML.

Ainsi les pages qui sont filtrées ne nécessitent pas que nous les ayions produites, les filtres peuvent s'adapter à n'importe quelles pages et notamment aux pages non écrites en XML ce qui est un gage de réemploi de matériau ancien tel qu'il fut originellement écrit.

Le mécanisme actuel de spécification des filtres passe par l'écriture de programmes Java (pour des raisons de portabilité) aisément insérables dans le contexte d'un serveur local puis par la description, au moyen d'un fichier XML, des règles à considérer. Un outil graphique de mise en œuvre d'une batterie de filtres prédéfinis qu'il ne s'agirait plus que de paramétrer, est bien sûr envisageable.

Tous ces programmes [7] ont été utilisés pour notre cédérom. Ils sont libres d'emploi et disponibles, dans leur état courant, sous licence GPL (*Gnu Public License*).

8. Bibliographie

[BRU 96] BRUSILOVSKY P., SCHWARZ E., WEBER G., « ELM-ART : An intelligent tutoring system on World Wide Web », FRASSON C., GAUTHIER G., LESGOLD A., Eds., *Intelligent Tutoring Systems*, vol. 1086 de *Lecture Notes in Computer Science*, Springer-Verlag, 1996, p. 261–269.

[CAZ 00a] CAZES C., QUEINNEC C., STEINBERG C., « Enseignement du langage C à l'aide d'un cédérom et d'un site – Mise en œuvre et observation », *Colloque international – Technologie de l'Information et de la Communication dans les Enseignements d'ingénieurs et dans l'industrie – TICE 2000*, Troyes (France), 2000, p. 63–63, version courte de [CAZ 00b].

[CAZ 00b] CAZES C., QUEINNEC C., STEINBERG C., « Enseignement du langage C à l'aide d'un cédérom et d'un site – Mise en œuvre et observation », Troyes (France), Atelier TICE, 2000.

[COW 00] COWARD D., « Java™ Servlet Specification, v2.3 pre-FCS », SUN Microsystems, 2000.

[GOO 99] GOOSSENS M., RAHTZ S., *The LaTeX Web Companion – Integrating TeX, HTML and XML*, Addison-Wesley, 1999.

[MAR 99] MARANGET L., « Hevea, un traducteur de LaTeX vers HTML en Caml », *Actes des 10èmes Journées Francophones des Langages Applicatifs*, INRIA, 1999, http ://pauillac.inria.fr/~maranget/hevea/.

[MOU 00] MOULIN C., PAZZAGLIA J.-C., « Création dynamique d'activités adaptées dans un environnement d'apprentissage à distance », *Technologies de l'Information et de la Communication dans les Enseignements d'ingénieurs et dans l'industrie – Colloque international*, Troyes (France), 2000, p. 179–183.

[PEL 00] PELEGRÍ-LLOPART E., « JavaServer Pages™ Specification, version 1.2 PD1 », SUN Microsystems, 2000.

[QUE 00] QUEINNEC C., « Enseignement du langage C à l'aide d'un cédérom et d'un site – Architecture logicielle », *Colloque international – Technologie de l'Information et de la Communication dans les Enseignements d'ingénieurs et dans l'industrie – TICE 2000*, Troyes (France), 2000, CNED, p. 93–102.

[SOM 00] SOMMARUGA L., CATENAZZI N., GIROUX S., MOULIN C., « A Distance Learning Environment Architecture », *Technologies de l'Information et de la Communication dans les Enseignements d'ingénieurs et dans l'industrie – Colloque international*, Troyes (France), 2000, p. 103–111.

7. `http://youpou.lip6.fr/queinnec/VideoC/`